On the Computational Complexity of MaxSAT

Mohamed El Halaby
Department of Mathematics
Faculty of Science
Cairo University
Giza, 12613, Egypt
halaby@sci.cu.edu.eg

Abstract

Given a Boolean formula in Conjunctive Normal Form (CNF) $\phi = S \cup H$, the MaxSAT (Maximum Satisfiability) problem asks for an assignment that satisfies the maximum number of clauses in ϕ. Due to the good performance of current MaxSAT solvers, many real-life optimization problems such as scheduling can be solved efficiently by encoding them into MaxSAT. In this survey, we discuss the computational complexity of MaxSAT in detail in addition to major results on the classical and parameterized complexities of the problem.

1 Introduction and Preliminaries

This paper surveys the computational complexity of the MaxSAT problem. It consists of three sections and is structured as follows: The first section introduces the MaxSAT problem and the world of propositional logic and satisfiability. Section two discusses the computational complexity of MaxSAT from the classical perspective. Finally, the third section discusses the parameterized complexity of MaxSAT and its recent results.

The satisfiability problem (SAT), which is the problem of deciding if there exists a truth assignment that satisfies a Boolean formula in conjunctive normal form (CNF), is a core problem in theoretical computer science because of its central position in complexity theory. Given a Boolean formula, SAT asks if there is an assignment to the variables of the formula such that it is satisfied (evaluates to true). SAT was the first problem proven to be NP-complete by Cook [9]. Each instance of an NP-complete problem can be translated into an instance of SAT in polynomial time. This makes it very important to develop fast and efficient SAT solvers.

An important optimization of SAT is MaxSAT which asks for a truth assignment that satisfies the maximum number of clauses of a given CNF formula. Many theoretical and practical problems can be encoded into SAT and MaxSAT such as debugging [30], circuits design and scheduling of how an observation satellite captures photos of Earth [36], course timetabling [2, 24, 23, 22], software package upgrades [16], routing [38, 25], reasoning [31] and protein structure alignment in bioinformatics [28].

A *Boolean variable* x can take one of two possible values 0 (false) or 1 (true). A *literal* l is a variable x or its negation $\neg x$. A *clause* is a disjunction of literals, i.e., $\bigvee_{i=1}^{n} l_i$. A *CNF formula* is a conjunction of clauses [11]. Formally, a CNF formula ϕ composed of k clauses, where each clause C_i is composed of m_i is defined as

$$F = \bigwedge_{i=1}^{k} C_i$$

where

$$C_i = \bigvee_{j=1}^{m_i} l_{i,j}$$

In this paper, a set of clauses $\{C_1, C_2, \ldots, C_k\}$ is referred to as a Boolean formula. A truth assignment *satisfies* a Boolean formula if it satisfies every clause.

MaxSAT is a generalization of SAT. Given a CNF formula ϕ, the problem asks for a truth assignment that maximizes the number of satisfied clauses in ϕ. For example, if $F = \{(x \vee \neg y), (\neg x \vee z), (y \vee z), (\neg z)\}$ then $I = \{x = 0, y = 0, z = 0\}$ satisfies three clauses. In fact, ϕ is unsatisfiable and thus the maximum number of clauses that can be satisfied in ϕ is three.

Decision problems are one of the central concepts of computational complexity theory. A *decision problem* is a special type of computational problems whose answer is either $True$ or $False$.

The next is an example of a problem from the field of graph theory. Given a graph $G = (V, E)$, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges, a *clique* is a subset $C \subseteq V$ such that for every pair of distinct vertices there is an edge, i.e., for all $u, v \in C, (u, v) \in E$.

Example 1.1 (Clique)**.** *Input: A graph $G = (V, E)$ and a natural number k.*
Decision problem: Output $True$ if G has a clique C such that $|C| = k$, or $False$ otherwise.
Search problem: Find a clique C in G such that $|C| = k$ if one exists, else output $\emptyset$.

The following example illustrates the difference between two versions of the SAT problem: the decision version and the search version.

Example 1.2 (SAT)**.** *Input: A CNF formula ϕ.*
Decision problem: Output $True$ if ϕ is satisfiable, or $False$ otherwise.
Search problem: Find an assignment that satisfies ϕ if one exists, else output $\emptyset$.

If the search problem can be solved, then certainly, the decision version can be solved as well. For example, in SAT, given a CNF formula ϕ, if a satisfying assignment is found, the output of the decision version is $True$, if no satisfying assignment can be found, the decision version returns $False$. So, search is harder, and if the search version can be solved, then we can certainly solve the decision problem.

To prove that a problem is at least as hard as another problem, we need to use a tool called *reductions*.

Definition 1.1 (Reduction)**.** *We say that a problem A reduces to another problem B if there exists a procedure R which, for every instance x of A, produces an instance $R(x)$ of B, such that the answer to $R(x)$ is also the answer to x.*

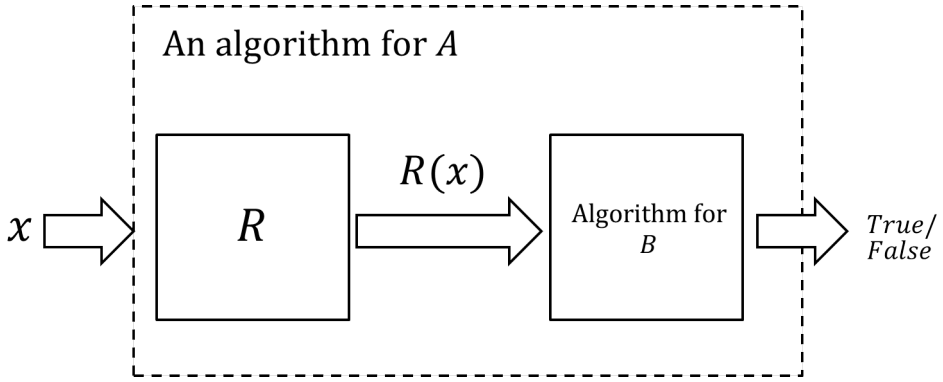

Figure 1: Reduction from A to B.

If it is possible to establish the scenario in figure 1, then it is reasonable to say that A is at least as hard as B. However, if we do not limit the amount of time required to compute R, this does not hold. R shall be a polynomial-time algorithm.

Next, we define the notion of complexity classes and the relation between these classes and reductions.

Definition 1.2 (Complexity class)**.** *A complexity class is a set of problems of related resource-based complexity. Typically, a complexity class is defined by: a model of computation, a resource (or collection of resources) and a function known as the complexity bound for each resource[3].*

Throughout this survey, the model of computation is the Turing machine and the resource we are interested in is time. However, the complexity classes defined later have different complexity bounds. Thus, a complexity class is basically a group of related resource-based complexity. The time resource is essentially how many steps are required to solve the problem. Another resource addressed by computational complexity is *space* (i.e., how much memory is required to solve the problem). The following are definitions of the fundamental complexity classes in complexity theory.

Definition 1.3 (Class P)**.** *The set of decision problems solvable in polynomial time.*

An example of a P problem is checking whether a number is prime[1].

Definition 1.4 (Class NP)**.** *The set of decision problems for which an answer can be verified in polynomial time.*

It is easy to see that $P \subset NP$, since getting the answer for a P problem is done in polynomial time.

Definition 1.5 (Class NP-complete)**.** *A decision problem A is said to be NP-complete if it is in NP, and for every problem H in NP, there is a polynomial-time reduction from H to A.*

Definition 1.6 (Class NP-hard)**.** *A decision problem A is said to be NP-hard if there is a polynomial-time reduction from an NP-complete problem A' to A[34]. Equivalently, a problem A is NP-hard when every problem L in NP can be reduced in polynomial time to A.*

An easy way to prove that a problem is NP-complete is first to prove that it is in NP, and then to reduce some known NP-complete problem to it. The difference between the two classes NP-hard and NP-complete is that, for a problem to be NP-complete, it must be in NP. However, an NP-hard problem need not be in NP. The diagram in figure 2 illustrates the differences between the four complexity classes.

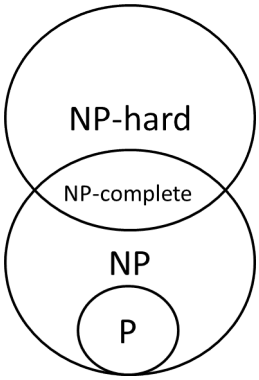

Figure 2: Diagram for P, NP, NP-complete and NP-hard in case P≠NP.

3SAT was the first known example of an NP-complete problem[9]. This implies that there is no algorithm that solves the problem in polynomial time. Other examples of NP-complete problems are Clique and Knapsack. Later, we will present a reduction from 3SAT to Max-2-SAT in order to prove the NP-hardness of Max-2-SAT.

Definition 1.7 (Max-2-SAT)**.** *Given a 2-CNF $\phi = \{C_1, \ldots, C_m\}$ and a positive integer $k \leq m$, the Max-2-SAT problem asks whether there is a truth assignment to the variables of ϕ that satisfies k or more clauses[13].*

Example 1.3. *The Max-2-SAT instance $\{(x), (y), (\neg z), (w), (\neg x \vee \neg y), (\neg y \vee z), (\neg x \vee z), (x \vee \neg w), (y \vee \neg w), (\neg z \vee \neg w)\}, k = 7$ has the answer $True$. Indeed, the assignment $\{x = True, y = False, z = False, w = True\}$ satisfies 7 clauses.*

An example of an NP-hard problem is *subset sum*, which is given a set o integers, checking whether or not there is a non-empty subset whose sum is zero.

In the following section, we address the complexity of Max-2-SAT and establish that it is NP-complete. The reason for discussing Max-2-SAT is that it is the most basic MaxSAT problem and thus what holds for Max-2-SAT will certainly hold for more general MaxSAT formalisms. In addition, the number of occurrences of the each variable in the input formula contributes to the complexity of MaxSAT and thus, we address a MaxSAT formalism called (k, s)-MaxSAT, which is the MaxSAT problem on formulae with k variables in each clause and each variable occurs s times.

Finally, section 3 addresses the parameterized complexity of MaxSAT. In this section, we address the parameterized complexity of MaxSAT in subsection 3.1 and show that it is fixed parameter tractable. Following that, the parameter is set above guaranteed values on the solution and the complexity of parameterized MaxSAT is readdressed in subsection 3.2. In addition, if n is the number of variables and m is the number of clauses in a formula ϕ, if the number of literals in each clause is bounded above by a function $r(n)$ in the number of variables, the corresponding MaxSAT formalism is called Max-$r(n)$-SAT. The tractability of Max-$r(n)$-SAT depends on $r(n)$ and this is addressed in subsection 3.3. Finally, a recent bound introduced by Bliznets and Golovnev[5] on parameterized MaxSAT is described in subsection 3.4.

2 Results and Discussion in Classical Complexity

2.1 NP-completeness of Max-2-SAT

Even though we are concerned with WPMaxSAT, it is true that MaxSAT is the simplest variation among all the MaxSAT formalisms mentioned. Also, the most basic MaxSAT instances are those having at most two literals in each clause, which are referred to as Max-2-SAT. Hence, proving that Max-2-SAT is NP-hard implies that the other formulations are at least NP-hard. Note that proving that a problem is $NP - complete$ also means that it is NP-hard (see figure 2).

Theorem 2.1. *Max-2-SAT is NP-complete.*

Proof. First, we will show that Max-2-SAT is in NP. This is obvious, because checking whether an assignment satisfies a certain number of clauses or not can be done in polynomial time.

In order to prove NP-hardness, we will reduce the 3SAT problem to Max-2-SAT[27]. Once that is done, then we can say that Max-2-SAT is NP-complete, since its solution leads to the solution of 3SAT, which is NP-complete. Recall that in 3SAT, we are given a 3-CNF formula and asked to find a satisfying assignment for it. For a 3SAT instance ϕ, we will convert it into a Max-2-SAT instance $R(\phi)$, clause by clause.

Given a 3SAT formula ϕ with m clauses, for each clause $C_i = (x \vee y \vee z) \in \phi$, where x, y and z are literals, we will introduce a new variable w_i (different from all the variables in ϕ) for this clause. We then replace C_i by the following 10 clauses:

$$(x), (y), (z), (w_i)$$
$$(\neg x \vee \neg y), (\neg y \vee \neg z), (\neg x \vee \neg z)$$
$$(x \vee \neg w_i), (y \vee \neg w_i), (z \vee \neg w_i)$$

We replace each clause in the 3SAT formula by 10 clauses in this manner, introducing a fresh variable w_i for every clause C_i. We end up with a 2-CNF formula, $R(\phi)$, with $10m$ clauses. We now claim that:

1. If an assignment satisfies $(x \vee y \vee z)$, then there is an assignment for w_i, that together with the assignment for x, y and z satisfies exactly 7 of the 10 clauses above.

2. If an assignment falsifies $(x \vee y \vee z)$, then every assignment for w_i, together with the assignment for x, y and z satisfies at most 6 of the 10 clauses above.

These claims imply that if there is a satisfying assignment for the original 3SAT formula, then there is an assignment for the new 2-CNF formula that satisfies at least $7m$ of the clauses. On the other hand, if no assignment satisfies the 3SAT formula, then at least for one of the original clauses, we will only be able to satisfy 6 of the 10 new clauses, and therefore, the number of clauses satisfied in the 2-CNF formula is strictly less than $7m$. Therefore, we can decide whether or not the given 3-CNF formula is satisfiable by determining whether or not it is possible to satisfy $7m$ of the clauses in the 2-CNF formula. It remains to prove the above claims, which we do by case analysis.

Case 1 $x = y = z = True$. Then by setting $w_i = True$, we can satisfy the entire first and third rows, or 7 clauses in all.

Case 2 $x = y = True$ and $z = False$. Then by setting $w_i = True$, we can satisfy three clauses from the first row, two from the second row, and two from the third row, or 7 clauses in all.

Case 3 $x = True$ and $y = z = False$. Then by setting $w_i = False$, we can satisfy one clause on the first row, and the entire second and third rows, which is again 7 clauses in all.

Case 4 $x = y = z = False$. Note that $(x \vee y \vee z)$ is not satisfied. If we set $w_i = True$, this only satisfies one clause on the first row, and all clauses on the second row; a total of 4 clauses. On the other hand if we set $w_i = False$, then we only satisfy all clauses on the second and third rows, a total of 6 clauses.

Other cases are analogous to the above. So, given a 3SAT instance $\phi = \{(x_1 \vee y_1 \vee z_1), \ldots, (x_m \vee y_m \vee z_m)\}$, where x_i, y_i and $z_i (1 \leq i \leq m)$ are literals, the corresponding Max-2-SAT instance is $(R(\phi), k)$, where

$$R(\phi) = \bigcup_{i=1}^{m} \{(x_i), (y_i), (b_i), (w_i),$$

$$(\neg x_i \vee \neg y_i), (\neg x_i \vee \neg z_i), (\neg y_i \vee \neg z_i),$$

$$(x_i \vee \neg w_i), (y_i \vee \neg w_i), (z_i \vee \neg w_i)\}$$

and

$$k = 7m$$

□

Example 2.1. *Let $\phi = \{(x \vee \neg y \vee \neg z), (\neg x \vee \neg y \vee \neg z)\}$ be a 3SAT instance, with $m = 2$ clauses. After reducing ϕ to Max-2-SAT, the generated clauses due to the first clause are*

$$(x), (\neg y), (\neg z), (w_1)$$

$$(\neg x \vee y), (y \vee z), (\neg x \vee z)$$

$$(x \vee \neg w_1), (\neg y \vee \neg w_1), (\neg z \vee \neg w_1)$$

and those due to the second clause are

$$(\neg x), (\neg y), (\neg z), (w_2)$$

$$(x \vee y), (y \vee z), (x \vee z)$$

$$(\neg x \vee \neg w_2), (\neg y \vee \neg w_2), (\neg z \vee \neg w_2)$$

Thus, $R(\phi) = \{(x), (\neg y), (\neg z), (w_1), (\neg x \vee y), (y \vee z), (\neg x \vee z), (x \vee \neg w_1), (\neg y \vee \neg w_1), (\neg z \vee \neg w_1), (\neg x), (\neg y), (\neg z), (w_2), (x \vee y), (y \vee z), (x \vee z), (\neg x \vee \neg w_2), (\neg y \vee \neg w_2), (\neg z \vee \neg w_2)\}$. *Running a MaxSAT solver on* $R(\phi)$ *gives the solution* $A = \{x = True, y = False, z = False, w_1 = True, w_2 = False\}$, *which satisfies exactly 7 clauses from the first group and 7 clauses from the second group,* $14 = 7(2) = 7m$ *in total. Thus, we can conclude that* ϕ *is satisfiable.*

2.2 (k, s)-MaxSAT

Definition 2.1 ((k, s)-formula)**.** *A CNF formula is called* (k, s) *if every clause has exactly* k *variables and every variable appears in at most* s *clauses.*

Definition 2.2 ($(\leq k, s)$-formula)**.** *A CNF formula is called* $(\leq k, s)$ *if every clause has at most* k *variables and every variable appears in at most* s *clauses.*

Definition 2.3 ((k, s)-MaxSAT)**.** *(k, s)-MaxSAT ($(\leq k, s)$-MaxSAT) is the MaxSAT problem where the input formula is (k, s) ($(\leq k, s)$).*

An important question is what is the minimum s for which $(2, s)$-MaxSAT is NP-complete. In[29] Raman, Ravikumar and Rao showed that $(\leq 2, 3)$-MaxSAT is NP-complete by first proving that $(\leq 2, 4)$-MaxSAT is NP-complete then forming a reduction from $(\leq 2, 4)$-MaxSAT to $(\leq 2, 3)$-MaxSAT.

Jaumard and Simeone[17] proved the first result (lemma 2.1) needed to prove the NP-completeness of $(\leq 2, 3)$-MaxSAT.

Definition 2.4 (Vertex cover)**.** *Given a graph $G = (v, E)$, a vertex cover is a subset V' of V such that for all edges $(u, v) \in E$ we have $u \in V', v \in V'$ or both. The subset V' is said to cover all the edges of G. Given a graph G and an positive integer k, the vertex cover problem asks whether or not G has a vertex cover of size at most k.*

Vertex cover is a classical NP-complete problem. Its NP-completeness can be proven by a reduction from 3SAT or from clique[18]. Figure 3 shows an example of a vertex cover.

Lemma 2.1. *$(\leq 2, 4)$-MaxSAT is NP-complete.*

Proof. The reduction is from vertex cover. It is clear that the problem is in NP. Given a cubic[1] graph $G = (V, E)$ such that $|E| = m$ and $|V| = n$, the MaxSAT instance $R(G)$ is constructed in the following steps:

1. For each vertex $v_i \in V$, introduce a Boolean variable x_i.
2. Add the unit clause (x_i) to $R(G)$.

[1] A cubic graph is a graph where the number of edges incident to each vertex is three. Vertex cover is also NP-compete for cubic graphs[14].

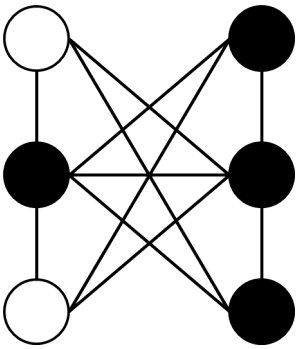

Figure 3: The vertices of the vertex cover are coloured black.

3. For each edge $(v_i, v_j) \in E$, add the clause $(\neg x_i \vee \neg x_j)$ to $R(G)$.

Now, $R(G) = \{(x_1), \ldots, (x_n)\} \cup \{(\neg x_i \vee \neg x_j) \mid (v_i, v_j) \in E\}$, having $n + m$ clauses with each clause having at most two literals. A variable x_i can appear in the clauses $(x_i), (\neg x_i \vee \neg x_{i_1}), (\neg x_i \vee \neg x_{i_2})$ and $(\neg x_i \vee \neg x_{i_3})$, which corresponds to a vertex appearing in three edges $(v_i, v_{i_1}), (v_i, v_{i_2})$ and (v_i, v_{i_3}), where v_{i_1}, v_{i_2} and v_{i_3} are vertices in V. So, each variable appears at most four times in $R(G)$, which means $R(G)$ is a $(\leq 2, 4)$-formula.

It remains to show that if G has a vertex cover of size at most k, then at least $n + m - k$ clauses in $R(G)$ can be satisfied. And, conversely, if there is an assignment for the variables of $R(G)$ that satisfies at least $n + m - k$ clauses, then G has a vertex cover of size at most k.

If G has a vertex cover of size at most k, then assigning the variables corresponding to the vertices of the vertex cover the value $False$ would satisfy at most k clauses in $R(G)$. This is because if a vertex v_i is in the vertex cover, then after setting x_i to $False$, all the clauses corresponding to edges in which v_i participates are satisfied.

Suppose $R(G)$ has an assignment that satisfies at least $n + m - k$ clauses. Without loss of generality, we can assume that the falsified clauses are the clauses corresponding to the vertices. To see this, consider an edge $\{v_i, v_j\} \in E$, where both x_i and x_j are $True$. This means the clause $(\neg x_i \vee x_j)$ is falsified. We can set one of x_i and x_j to $False$ in order to satisfy $(\neg x_i \vee \neg x_j)$ and falsify either (x_i) or (x_j), thus keeping at least $n + m - k$ clause satisfied. It is now clear that the vertices corresponding to the variables assigned $False$ construct a vertex cover of size at most k. □

Next, the previous lemma is used to prove $(\leq 2, 3)$-MaxSAT is NP-complete. The following notations are used in the proof. Given a formula ϕ

1. $Var(\phi)$ denote the set of variables appearing in ϕ.
2. For a variable $x \in Var(\phi)$, $d(x)$ denotes the number of clauses in which x appears.
3. The number of variables x for which $d(x)$ is 4 is denoted by b.

Theorem 2.2. *$(\leq 2, 3)$-MaxSAT is NP-complete.*

Proof. This can be shown via a reduction from $(\leq 2, 4)$-MaxSAT. Given a $(\leq 2, 4)$ formula ϕ, construct a formula $R(\phi)$ by doing the following steps for each $x \in Var(\phi)$ with $d(x) = 4$:

1. Replace the ith occurrence of x with a new variable $x_i, (1 \leq i \leq 4)$.

2. Add the following set of 16 clauses

$$\{(x_1 \vee y_1), (x_2 \vee y_2), (x_3 \vee y_3), (x_4 \vee y_4),$$
$$(\neg x_1 \vee \neg y_7), (\neg x_2 \vee \neg y_8), (\neg x_3 \vee \neg y_5), (\neg x_4 \vee \neg y_6),$$
$$(\neg y_1 \vee y_5), (\neg y_2 \vee y_6), (\neg y_3 \vee y_7), (\neg y_4 \vee y_8),$$
$$(\neg y_1 \vee y_5), (\neg y_2 \vee y_6), (\neg y_3 \vee y_7), (\neg y_4 \vee y_8),$$
$$(\neg y_1 \vee y_2), (\neg y_3 \vee y_4), (y_5 \vee \neg y_6), (y_7 \vee \neg y_8)\}$$

   where $y_1, \ldots, y_8$ are new variables. Let X be the accumulated set of $16b$ clauses added for all such variables x.

After performing the previous two steps, each $x \in Var(R(\phi))$ appears at most three times. That is, each $x_i, (1 \leq i \leq 4)$ appears once as a replacement for some occurrence of the variable x and twice in X. Each variable $y_i, (1 \leq i \leq 8)$ occurs three times in X. To complete the reduction, we will show that: there is an assignment that satisfies at least k clauses of ϕ **if and only if** there is an assignment that satisfies at least $k + 16b$ clauses of $R(\phi)$.

Assume ϕ has an assignment that satisfies at least k clauses. We can satisfy $k + 16b$ clauses in $R(\phi)$ by

1. Setting the truth values of $x_i, (1 \leq i \leq 4)$ to that of x, for every x with $d(x) = 4$.

2. Setting each $y_i, (1 \leq i \leq 8)$ to $False$ if x_i is $True$, or to $True$ if x_i is $False$.

The first step satisfies k clauses and the second satisfies the $16b$ clauses of X. In total, $k + 16b$ clauses are satisfied.

Conversely, assume $R(\phi)$ has an assignment that satisfies at least $k + 16b$ clauses. First, We will show how to modify this assignment such that it satisfies all the $16b$ clauses of X while still satisfying at least $k + 16b$ clauses. Next, we will show that the only assignments that satisfy all the clauses of X are the ones in which the values for the x_i variables and the value of x are the same (i.e., either all are $True$ or all are $False$), and from that we can construct an assignment for ϕ.

Consider $x_i, (1 \leq i \leq 4)$ variables in the 16 clauses above. There are three cases regarding their truth values:

1. All $x_i, (1 \leq i \leq 4)$ are $True$ or all are $False$. In this case all the 16 clauses are satisfiable.

2. Exactly one the x_i variables is $False$ or exactly one is $True$. In this case, 15 clauses are satisfiable. To see this, let (without loss of generality)

- $x_1 = True$ and $x_2 = x_3 = x_4 = False$. After plugging in these values, we are left with the following unsatisfiable set of clauses:

$$\{(y_2), (y_3), (y_4),$$
$$(\neg y_7),$$
$$(\neg y_1 \vee y_5), (\neg y_2 \vee y_6), (\neg y_3 \vee y_7), (\neg y_4 \vee y_8),$$
$$(\neg y_1 \vee y_2), (\neg y_3 \vee y_4), (y_5 \vee \neg y_6), (y_7 \vee \neg y_8)\}$$

  At most, we can satisfy all except one of them, thus satisfying 15 clauses in total. Setting $y_i, (1 \leq i \leq 8)$ to $True$ satisfies all the 16 clauses except $(\neg x_1 \vee \neg y_7)$.

- $x_1 = False$ and $x_2 = x_3 = x_4 = True$. After plugging in these values, we are left with the following unsatisfiable set of clauses:

$$\{(y_1),$$
$$(\neg y_8), (\neg y_5), (\neg y_6),$$
$$(\neg y_1 \vee y_5), (\neg y_2 \vee y_6), (\neg y_3 \vee y_7), (\neg y_4 \vee y_8),$$
$$(\neg y_1 \vee y_2), (\neg y_3 \vee y_4), (y_5 \vee \neg y_6), (y_7 \vee \neg y_8)\}$$

  At most we can satisfy 15 clauses in total. Setting $y_i, (1 \leq i \leq 8)$ to $False$ satisfies all the 16 clauses except $(x_1 \vee \neg y_1)$.

  Thus, 15 clauses at most can be satisfied in both cases.

3. Exactly two of the x_i variables are $True$. In this case, at most 14 clauses are satisfiable. We have two subcases:

   - $x_1 = x_2 = True$ and $x_3 = x_4 = False$. This assignment falsifies a subset of three clauses[2] (independently) in X, such as $\{(x_3 \vee y_3), (\neg y_3 \vee y_7), (\neg y_7 \vee \neg x_1)\}$ or $\{(x_4 \vee y_4), (\neg y_4 \vee y_8), (\neg y_8 \vee \neg x_2)$. However, setting every $y_i, (1 \leq i \leq 8)$ to $False$ leads to only two unsatisfied clauses in X instead of three, namely $(x_3 \vee y_3)$ and $(x_4 \vee y_4)$. The same argument works if $x_1 = x_2 = False$ and $x_3 = x_4 = True$.
   - $x_1 = x_3 = True$ and $x_2 = x_4 = False$. This assignment falsifies a subset of four clauses (independently) in X, such as $\{(x_2 \vee y_2), (\neg y_2 \vee y_6), (\neg y_6 \vee y_5), (\neg y_5 \vee \neg x_3)\}$ or $\{(x_4 \vee y_4), (\neg y_4 \vee y_8), (\neg y_8 \vee y_7), (\neg y_7 \vee \neg x_1)\}$. However, setting every $y_i, (1 \leq i \leq 8)$ to $True$ satisfies all the clauses of X, except $(\neg x_1 \vee y_7)$ and $(\neg x_3 \vee y_5)$. The case where $x_1 = x_2 = True$ and $x_3 = x_4 = False$ is symmetric.

Hence, the only two assignments that satisfy every clause in X are the ones where all the values of $x_i, (1 \leq i \leq 4)$ are the same (i.e., all are $True$ or or all are $False$).

Starting with an assignment for $R(\phi)$ that satisfies at least $k + 16b$ clauses, in order to obtain an assignment that satisfies all the $l6b$ new clauses and still satisfy $k + 16b$ clauses in total, do the following for each set X of the 16 new clauses.

[2]Not all the clauses can be falsified together.

1. If the values of $x_i, (1 \leq i \leq 4)$ are all the same (the first case) then we are done.

2. If exactly one of the $x_i, (1 \leq i \leq 4)$ variables is either $True$ or $False$ (the second case), then by flipping the value of that variable (either from $True$ to $False$ or from $False$ to $True$) we will satisfy one extra clause and we may falsify at most one original clause, in particular the clause containing that variable.

3. If exactly two of the $x_i, (1 \leq i \leq 4)$ variables are $True$ (the third case), then we flip the values of these two variables (from $True$ to $False$) to satisfy two extra clauses in X and falsify at most two original clauses, in particular the two clauses containing these two variables.

□

Next, we will use the fact that $(\leq 2, 3)$-MaxSAT is NP-complete to show that $(2, 3)$-MaxSAT is also NP-complete.

Theorem 2.3. *(2,3)-MaxSAT is NP-complete.*

Proof. The reduction is from $(\leq 2, 3)$-MaxSAT. Given a $(\leq 2, 3)$-MaxSAT formula ϕ with m clauses, replace each unit clause (x_i) with the two clauses $(x_i \vee y_i)$ and $(x_i \vee \neg y_i)$, where y_i is a new variable. By doing so, we have changed ϕ into a new instance $R(\phi)$ where each clause has exactly two literals.

If the number of unit clauses in ϕ is u, then $|R(\phi)| = m + u$ clauses. An assignment that satisfies at least k clauses in ϕ clearly satisfies at least $k + u$ clauses in $R(\phi)$. Also, if $R(\phi)$ has an assignment that satisfies at least $k + u$ clauses, then the same assignment satisfies at least k clauses in ϕ by ignoring the values for the $y_i, (1 \leq i \leq u)$ variables.

However, the variables appearing in the unit clauses may appear four times after this transformation, and thus $R(\phi)$ becomes a $(2, 4)$-formula. Fortunately, we can use the reduction presented in theorem 2.2 to convert $R(\phi)$ into an equivalent $(2, 3)$-formula. □

Given an $(\leq n, 2)$-MaxSAT formula, algorithm 1 solves the problem in $O(n)$, where n is the number of variables in the input formula.

Algorithm 1: MaxSAT(ϕ) Solves the $(\leq n, 2)$-MaxSAT problem ϕ

```
Input: An (≤ n, 2)-MaxSAT instance φ = U ∪ N having m clauses over n variables, where
       U is the set of unit clauses and N is the set of non-unit clauses
Output: An assignment that satisfies the maximum number of clauses in φ
1  foreach variable v of φ do
2      if v appears either only positively or only negatively then
3          set the value of v appropriately
4          remove the clauses in which v appears (from U and/or N)

       // At this point, the variables that are left unset appear twice, where one of the
   occurences is positive and the other is negative
5  while U ≠ ∅ do
6      choose a variable v that appears in a unit clause UC
7      set v in such a way to satisfy UC
8      U ← U \ UC
9      if v occurs in another clause C then
           // v appears in C in its complementary case
10         if C ∈ U then
11             U ← U \ C
12         else
               // C is in N
13             remove v from C
14             if C becomes a unit clause then
15                 U ← U ∪ C
16                 N ← N \ C
17 if N ≠ ∅ then
18     choose a clause C ∈ N
19     choose a literal y in C
20     set y to True
21     N ← N \ C
22     if y occurs in another clause C′ then
23         remove y from C′
24         if C′ becomes a unit clause then
25             U ← U ∪ C
26             N ← N \ C
27     MaxSAT(φ)                          // Recurse on φ after the previous changes
28 if there exists a variable v that is unset then
29     set v arbitrarily
```

The algorithm starts by looping over the variables of ϕ to identify those which occur only positively or only negatively. After identifying such variables, there values are set in such a way to satisfy the clauses in which they appear. Next, the satisfied clauses are removed from the formula (lines 1-4).

In each step of the while loop (lines 5-16) a unit clause is removed from U after satisfying it by setting its literal to $True$ (lines 6-8). If the variable appearing in the removed unit clause occurs negatively in another clause C, then C is removed if it is a unit clause (lines 9-11). Otherwise, the negative occurrence of the variable is removed from C (line 13). Now, if C becomes a unit clause, then it is added to U, then it is removed from N (line 14-18). The loop continues until all the unit clauses have been removed.

After the while loop ends, if N still contains any clauses (line 17), then one is picked (line 18) and one of its literals is chosen and set to $True$ (line 20) in order to satisfy the entire clause. Next, if the chosen literal occurs in another clause, then it is removed from that clause (lines 22-23). If the clause now becomes unit, it is added to U then removed from N (lines 24-26). Finally, a recursive call to the algorithm is made on the reduced formula (line 27).

The correctness of algorithm 1 is clear since a unit clause can be satisfied by

satisfying their literals. If there are no unit clauses, then any variable in one of the clauses in the formula can be set to satisfy the clause. In[32], Tovey showed that a CNF formula is always satisfiable if every clause contains more than one variable, and every variable appears once positively and once negatively. After algorithm 1 satisfies all the unit clauses, we are left with a formula satisfying the two previous properties.

The loop in lines 1-4 takes time linear in the number of variables n. This is because in one pass over ϕ, by keeping appropriate counters, we can identity unit clauses and those variables that appear either only positively or only negatively. Also, setting the value of a variable and removing the clauses in which it appears take constant time.

In lines 5-16, each step of the while loop removes at least one clause from ϕ and the loop terminates when no unit clauses remain. The assignment statements, setting the values of variables and adding or removing a clause from a set all take constant time in each step.

Consequently, the entire algorithm takes $O(|\phi|)$ time and since each variable appears at most twice, then $|\phi| \leq 2n$. So, the running time is $O(n)$.

The next theorem is a direct result from algorithm 1.

Theorem 2.4. *The $(\leq n, 2)$-MaxSAT problem is in class P.*

Proof. Given a $(\leq n, 2)$-MaxSAT formula, algorithm 1 can solve the instance in polynomial time. □

3 Results and Discussion in Parametrized complexity

The aim of standard computational complexity is to classify problems by the amount of resources needed to solve them. The fundamental idea of measuring the required amount of the resource as a function of the input size has led to a variety of complexity classes, such as the ones discussed previously.

However, leaning on developing a sense of intractability towards a particular problem just by calculating its complexity only in terms of its input size means ignoring any structure about the instance. This method can make some problem seem harder than they really are. The theory of *parametrized complexity* attempts to overcome this issue of overestimating the difficulty by measuring the complexity not only by the size of the input, but also in terms of a numerical parameter that depends on the input in some way.

The main idea in parametrized complexity is that it loosens the concept of tractability (i.e., polynomial-time solvability) by accepting algorithms whose running times are non-polynomial (usually exponential) in the parameter only, this is called *fixed-parameter tractability* (FPT). For example, the vertex cover problem has an algorithm whose running time is $O(kn + 1.2738^k)$[8], where k (the parameter) is the size of the vertex cover and n is the number of vertices in the graph. Notice that the running time in this example is linear in the number of vertices but exponential in the parameter. In other words, the algorithm gets exponentially slower as the size of the vertex cover (the solution) gets bigger.

Definition 3.1 (Parametrized problem)**.** *A parametrized problem* Π *is a collection of* $(input, parameter)$ *pairs, where input is an instance and parameter*

is a natural number.

Example 3.1. *In the following examples of parametrized problems, $G = (V, E)$ is a graph and ϕ is a CNF formula.*

- *$k-$Vertex cover: $\{(G, k) \mid$ there is a vertex cover in G of size $k\}$.*
- *$k-$Clique: $\{(G, k) \mid$ there is a clique in G of size $k\}$.*
- *SAT-VAR: $\{(\phi, k) \mid \phi$ is satisfiable and has k variables$\}$.*
- *SAT-LEN: $\{(\phi, k) \mid \phi$ is satisfiable and $|\phi| \leq k\}$.*
- *SAT-TRUE: $\{(\phi, k) \mid \phi$ is satisfiable with k variables set to $True\}$. It is not known whether this problem is in FPT.*

Definition 3.2 (Class FPT)**.** *A parameterized problem Π is said to be fixed-parameter tractable (or belongs to the class FPT) if there is an $O(n^a f(k))$ time algorithm that solves each instance (I, k) of Π, where n is the size of I, a is a constant independent of k and f is an arbitrary (computable, but could be exponential) function of k.*

The parametrized version of MaxSAT that we are concerned with in this section is: given a CNF formula ϕ and a parameter k, is there an assignment that satisfies at least k clauses in ϕ. In[6], Cai and Chen showed that Max-c-SAT[3] is FPT.

3.1 Parameterized MaxSAT is FPT

This subsection is concerned with proving that parameterized MaxSAT is FPT. Theorem 3.1 proves this result using lemmas 3.1 and 3.2.

Lemma 3.1. *Given a CNF formula ϕ having m clauses, there exists an assignment satisfying at least $\lceil \frac{m}{2} \rceil$ clauses.*

Proof. Choose any assignment A for ϕ. If A satisfies $\lceil \frac{m}{2} \rceil$ or more, then we are done. If not, then the bitwise complement assignment $\bar{A}$ of A satisfies every clause falsified by A and falsifies every clause satisfied by A. Thus, $\bar{A}$ satisfies at least $\lceil \frac{m}{2} \rceil$ clauses. □

Lemma 3.2. *If there are k clauses in a given CNF formula ϕ such that each of these k clauses have at least k literals, then in $O(|\phi|)$ time we can find an assignment that satisfies all these k clauses.*

Proof. Let $\phi_k = \{C_1, \ldots, C_k\}$ be a set of k clauses, each having at least k literals. To find an assignment that satisfies ϕ_k, do the following: at step i, set the first unassigned literal in clause C_i to $True$ and . We are guaranteed to find an unassigned literal at step i, since there are $i - 1$ variables have already been assigned in previous $i - 1$ steps. thus, in a single loop over ϕ, clauses with k or more literals can be identified and satisfied. Hence, the assignment can be found in $O(|\phi|)$ time. □

[3]Given a CNF formula ϕ such that each clause in ϕ has up to c literals, and a parameter k, find an assignment that satisfies at least k clauses in ϕ.

Algorithm 2 solves the MaxSAT parametrized problem.

Algorithm 2: FPT-MaxSAT(ϕ, k)

Input: An CNF formula ϕ with m clauses and a parameter k
Output: $True$, if there is an assignment that satisfies at least k clauses of ϕ, $False$ otherwise

Step 1 If $k > m$, then output $False$ and terminate.

Step 2 If $k \leq \lceil \frac{m}{2} \rceil$, then output $True$ and terminate.

Step 3 Let ϕ_L be the $\{C \mid C \in \phi$ and C has k or more literals$\}$ and $\phi_S \leftarrow \phi \setminus \phi_L$. Now, $\phi = \phi_L \cup \phi_S$. If $|\phi_L| \geq k$, then output $True$ and terminate.

Step 4 Construct a binary tree T, where each node is a pair (f, j) such that f is a formula and j is a non-negative integer. Build T in the following way:

1. Let the pair $(\phi_S, k - |\phi_L|)$ be the root of T.
2. For any node (f, j)
   (a) If $j > |f|$, then make (f, j) a leaf node with the label No.
   (b) If $j = 0$, then make (f, j) a leaf node with the label Yes.
   (c) For a variable v appearing in f, let v_T (v_F) be the number of clauses in f that contains the literal v ($\neg v$). If for each variable u appearing in f, we have $u_T = 0$ or $u_F = 0$, then (f, j) is a leaf node labeled Yes.
   (d) Otherwise, among all the variables v with $v_T > 0$ and $v_F > 0$, choose a variable x that appears the maximum number of times (i.e., $v_T + v_F$ is maximum).
   If v occurs twice in f (i.e., $v_T + v_F = 2$), then f is an $(\leq n, 2)$-formula and we can obtain the maximum number of clauses N that can be satisfied in f by running algorithm 1 (with a slight modification to compute N). If $N \geq j$, then (f, j) is a leaf node labeled Yes, otherwise (f, j) is a leaf node labeled No.
   If $v_T + v_F > 2$, then expand (f, j) to have $(f_T, j - v_T)$ and $(f_F, j - v_F)$ as its left and right child respectively, where f_v ($F_{\neg v}$) is the formula where the literal v ($\neg v$) is assigned $True$.
3. At any step of T's construction, if a leaf node is labeled Yes, then output $True$ and terminate.

Step 5 Output No and terminate.

We will now show that the algorithm is correct. The first step is a simple case. The second step follows from lemma 3.1. The third step follows from lemma 3.2. The fourth step attempts to satisfy $k - |\phi_L|$ clauses by assigning one variable the value $True$ in each satisfied clause. At most $k - |\phi_L|$ variables are assigned in this process, and some clauses in ϕ_L may become satisfied as well. So, ϕ_L has at most $|\phi_L|$ clauses left, each of length at least $k - (k - |\phi_L|) = |\phi_L|$. Lemma 3.2 can be applied to satisfy the remaining clauses in ϕ_L, which makes the total number of satisfied clauses $(k - |\phi_L|) + |\phi_L| = k$ as required.

Each node in the tree T has a Yes/No label. For a node (f, j), the label represents whether we can satisfy j clauses in f or not, if so, then it is labeled Yes, otherwise it is labeled No. If it is not yet known whether it is possible to satisfy j clauses, then (f, j) is not given a label, and the algorithm attempts to assign the variables of f. The idea is that if j clauses can be satisfied in f, then the partial assignment constructed along some path from (f, j) to a leaf node labeled Yes indeed does the job. If all the paths from (f, j) lead to leaf

nodes labeled No, then the fifth step is executed and all the leaves of the subtree rooted at (f, j) are labeled No.

The first three steps can clearly be carried out in linear time. In the fourth step, the processing time required to expand a node (f, j) or to figure out if it is a leaf is $O(|f|)$. If (f, j) is an internal node, then its two children are have integers strictly less than j and formulae of sizes less than $|f|$. Furthermore, since v occurs more than twice (since (f, j) is an internal node), then $v_T + v_F \geq 3$, and so at least one of v_T and v_F is greater than or equal to 2. Thus, at least one child node has an integer strictly less than $j - 1$. Hence, the size of T satisfies the Fibonacci recurrence $F(j) \leq 1 + F(j-1) + F(j-2)$, which has the solution $F(j) \leq \varphi^j$, where $\varphi = \frac{1+\sqrt{5}}{2} = 1.6180339\ldots$ is the golden ratio. Since the root node of T has the integer $k - |\phi_L|$, the number of nodes in T is bounded by the solution to $F(j)$ at $k - |\phi_L|$, which is $\varphi^{k-|\phi_L|}$.

So, T can be constructed in $O(|\phi_S|\varphi^{k-|\phi_L|})$ time. To obtain an upper bound for the running time of the algorithm, we need to bound $|\phi_S|$ from above. Since the fourth step (the construction of the binary tree) is only executed if $m < 2k$ (since $k > \lceil \frac{m}{2} \rceil$) and the root of the tree has integer $k - |\phi_L|$, then $|\phi_S| < cm < 2ck$, assuming ϕ is a c-SAT formula[4]. Thus, the entire algorithm has a running time of $O(|\phi| + ck\varphi^k)$.

Theorem 3.1. *Given a CNF formula ϕ and a non-negative integer k, algorithm 2 can find an assignment that satisfies at least k clauses in ϕ in $O(|\phi| + k^2 2^k)$ time.*

3.2 Parameterizing MaxSAT above guaranteed values

Parameterizations above guaranteed values is Parameterizations setting the parameter k to be a quantity above a lower bound or below an upper bound. For example, given a CNF formula with m clauses, a possible lower bound on the number of satisfied clauses is $\lceil \frac{m}{2} \rceil$. It is an interesting question to find out how difficult it is to satisfy a number of clauses that is beyond the lower bound. Theorem 3.2 shows that MaxSAT problem of determining if we can satisfy at least $\rceil \frac{m}{2} \lceil + k$ clauses is FPT. This result is proved using lemmas 3.3 and 3.1.

Lemma 3.3. *If each variable in a given CNF formula ϕ is set to $True$ or $False$ uniformly and independently, then the probability of satisfying a unit clause in ϕ is $\frac{1}{2}$ and the probability of satisfying a non unit clause in ϕ is at least $\frac{3}{4}$.*

Proof. A unit clause has either the form (x) or $(\neg x)$, where x is a variable. Since x is set uniformly and independently, then the probability a unit cluase is satisfied is $\frac{1}{2}$. The shortest non-unit clause has the form $C = (l_1 \vee l_2)$, where l_1 and l_2 are literals and both can either be a variable or the negation of a variable. It is easy to see that among all the four possible assignments for C, there is only one assignment that falsifies C, which is the one where both l_1 and l_2 are falsified. Thus, the probability that a non-unit clause is satisfied is at least $\frac{3}{4}$. □

Corollary 3.1. *Given a CNF formula ϕ with m clauses, of which p are non-unit, an assignment exists that satisfies at least $\lceil \frac{m}{2} \rceil + \frac{p}{4} - 1$ clauses, and it can be obtained in $O(|\phi|)$ time.*

[4]If every clause in a CNF formula has up to c clauses, then it is called a c-SAT formula.

Proof. If we set each variable in ϕ to $True$ or $False$ uniformly and independently, from lemma 3.3, the expected number of satisfied unit clauses and non-unit clauses by this random assignment is $\frac{m-p}{2}$ and $\frac{3p}{4}$ respectively. So, in total we have at least $\frac{m-p}{2} + \frac{3p}{4} = \frac{m}{2} + \frac{p}{4}$ clauses expcted to be satisfied. Thus, there is an assignment that satisfies at least $\lceil \frac{m}{2} + \frac{p}{4} \rceil \geq \lceil \frac{m}{2} \rceil + \frac{p}{4} - 1$ clauses. □

Theorem 3.2. *Given a CNF formula ϕ with m clauses and a non-negative integer k, we can find an assignment that satisfies at least $\lceil \frac{m}{2} \rceil + k$ clauses in ϕ or find out that such an assignment does not exist in $O(|\phi| + k^2 2^k)$ time.*

Proof. Algorithm 3 accomplishes this task. If the number of unit clauses is zero, then all the clauses are unit and the algorithm identifies the variables with more positive occurrences and assigns them $True$ (lines 3-4) and assigns the remaining variables $False$ (lines 3-5). If the number of non-unit clauses exceeds $4k+4$, then lemma 3.1 can be applied to obtain an assignment that satisfies the required number of clauses, since $\lceil \frac{m}{2} \rceil + \frac{4k+4}{4} - 1 = \lceil \frac{m}{2} \rceil + k$ (lines 6-7). Each iteration of the while loop (lines 9-11) identifies two clauses of the form (x) and $(\neg x)$, then removes them from U. At the end of the while loop, if $|U| \geq \lceil \frac{m}{2} \rceil + k$, then the required number of clauses that needs to be satisfied can be reached since all the clauses left in U can be satisfied. Otherwise, algorithm 2 is called with the input ϕ (the reduced version) and k.

It is easy to see that the running time of the statements in lines 2-9 is $O(|\phi|)$. The final step calls algorithm 2 when $|U| \leq \lceil \frac{m}{2} \rceil + k - 1$ and $|N| \leq 4k + 4 - 1$. So, we have $m = |U| + |N| \leq \lceil \frac{m}{2} \rceil + k - 1 + 4k + 3$. Thus, $m \leq \lceil \frac{m}{2} \rceil + 5k + 2$, and hence the number of clauses required to be satisfied ($\lceil \frac{m}{2} \rceil + k$) is bounded by $6k + 3$. The running time is $O(|\phi| + k^2 \varphi^{6k})$, which follows from the time analysis of algorithm 2. □

In 2011, Gutin *et. al*[15] proved a similar result. They showed that if for any three clauses in a CNF formula ϕ there is an assignment that satisfies all of them (also known as 3-satisfiability), then the following parameterized MaxSAT problem is in FPT: determine whether there is an assignment that satisfies at least $\frac{2m}{3} + k$ clauses in ϕ, where $m = |\phi|$ and k is the parameter. In order to prove this result, they used an earlier theorem[20] that states that we can satisfy at least $\frac{2}{3}$ of the clauses of every 3-satisfiable formula.

Algorithm 3: FPT-MaxSAT(ϕ, k)

Input: An CNF formula ϕ with m clauses and a parameter k
Output: $True$, if there is an assignment that satisfies at least $S = \lceil \frac{m}{2} \rceil + k$ clauses of ϕ, otherwise $False$

```
1  T ← 0
2  let U be the set of unit clauses and N be the set of non-unit clauses
   // Now, we have φ = U ∪ N
3  if N = ∅ then
4      assign those variables with more positive occurrences the value True, and
       assign the remaining variables the value False
5      increment T whenever a clause is satisfied
6      if T ≥ k then
7          return True
8  else if |N| ≥ 4k + 4 then
9      return True
10 else
11     while U contains unit clauses (x) and (¬x) do
12         U ← U \ {(x), (¬x)}
13         T ← T + 1
14     if |U| ≥ ⌈m/2⌉ + k then
15         return True
16     return FPT-MaxSAT(φ,T)
```

3.3 Parameterized complexity of Max-$r(n)$-SAT

Now, we will discuss a special case of parameterized MaxSAT, called Max-$r(n)$-SAT, which is not in FPT.

Before defining the Max-$r(n)$-SAT problem, first note that a if each clause in a CNF formula ϕ with $|\phi| = m$ has $r_i, (1 \leq i \leq m)$ literals, then the expected number of clauses satisfied in ϕ by a random assignment (each variable is set to either $True$ or $False$ uniformly and independently) is $asat(\phi) = (1 - \frac{1}{2^{r_1}}) + \cdots + (1 - \frac{1}{2^{r_m}}) = \sum_{i=1}^{m}(1 - \frac{1}{2^{r_i}})$. This follows from the fact that the the only assignment (among all the possible 2^{r_i} assignments) that falsifies a clause with r_i literals is that which falsifies all the r_i literals.

Definition 3.3 (Max-$r(n)$-SAT)**.** *Given a CNF formula ϕ with m clauses over n variables such that each clause has at most $r(n)$ literals, and a parameter k, the Max-$r(n)$-SAT problem asks whether or not there is an assignment that satisfies at least $asat(\phi) + k$ clauses.*

Unlike c-SAT instances where each clause has at most c literals, for a fixed constant c, the number of literals in each clause of a Max-$r(n)$-SAT formula is a function in the number of variables. This change takes MaxSAT out of the FPT class and into a more difficult class called para-NP-complete (see theorem 3.3).

Definition 3.4 (para-NP)**.** *A parameterized problem Π is in the class para-NP if a pair (I, k) can be decided whether or not it is in Π in $O(n^a f(k))$ time, where $n = |I|$, f is a function of k only and a is a constant independent from k.*

Definition 3.5 (FPT reduction)**.** *Given two parameterized problems* Π_1 *and* Π_2*, an FPT-reduction* R *from* Π_1 *to* Π_2 *is a many-to-one transformation from* Π_1 *to* Π_2*, such that*

1. $(I_1, k_1) \in \Pi_1$ *if and only if* $R((I_2, k_2)) \in \Pi_2$*, and* $k_2 \leq g(k_1)$ *for a fixed function* g*.*

2. R *can be computed in* $O(f(k)|I_1|^a)$ *time.*

Definition 3.6 (Class para-NP-complete)**.** *A parameterized problem* Π *is in the class para-NP-complete if*

1. Π *is in para-NP, and*

2. *for any parameterized problem* Π' *in para-NP, there is an FPT-reduction from* Π' *to* Π*.*

Definition 3.7 (Class XP)**.** *A parameterized problem* Π *is in the class XP if each instance* (I, k) *of* Π *can be be solved by an* $n^{O(f(k))}$ *time algorithm, where* $f(k)$ *is an arbitrary function of* k *only and* n *is the size of* I*.*

It is known that FPT $\subset$ XP[12]. In [12], Flum and Grohe showed that a para-NP problem Π is in para-NP-complete if there is a reduction from an NP-complete problem to the subproblem of Π, where the parameter is a fixed constant.

Theorem 3.3. *Max-*$r(n)$*-SAT is para-NP-complete for* $r(n) = \lceil \log n \rceil$*.*

Proof. It is easy to see that Max-$r(n)$-SAT is in para-NP. This is because we decide whether a given assignment satisfies $asat(\phi) + k$ clauses in polynomial time. To prove its para-NP-completeness, Crowston *et al.* [10] gave a reduction from a problem called *Linear-3-SAT* to Max-$r(n)$-SAT with k fixed to 2.

Given a 3-SAT formula ϕ with m clauses and n variables such that $m < cn$ (the number of clauses is linear in the number of variables), for a fixed constant c, Linear-3-SAT asks if there is an assignment that satisfies ϕ. Tovey showed in [33] that any 3-SAT is NP-complete, even for 3-CNF formulae with every variable appearing in at most four clauses. As result, the NP-completeness of Linear-3-SAT follows.

Let $\phi = \{C_1, \ldots, C_m\}$ be a Linear-3-SAT over the variables $x_1, \ldots, x_n$. We have $m < cn$, for a positive constant c, since ϕ is an instance to Linear-3-SAT. Construct a Max-$r(n)$-SAT formula $R(\phi)$ with $m' = 2^{\lceil \log n' \rceil + 1}$ clauses over the $n' = 2cn$ variables $x_1, \ldots, x_n, y_1, \ldots, y_{n'-n}$. The set of clauses of $R(\phi)$ is divided into three groups:

1. $G_1 = \{(l_1 \vee \cdots \vee l_{\lceil \log n' \rceil}) \mid l_i \in \{y_i, \neg y_i\}, 1 \leq i \leq \lceil \log n' \rceil\} \setminus \{\neg y_1 \vee \cdots \vee \neg y_{\lceil \log n' \rceil}\}$.

2. $G_2 = \{C_i \vee \neg y_4 \vee \cdots \vee \neg y_{\lceil \log n' \rceil} \mid 1 \leq i \leq m\}$.

3. G_3 consists of $m' - (|G_1| + |G_2|)$ clauses of length $\lceil \log n' \rceil$ over the variables $y_{\lceil \log n' \rceil + 1}, \ldots, y_{n'-n}$, such that each variable appears in at least one clause and each clause contains only positive literals.

Every clause in $R(\phi)$ (belonging to either G_1, G_2 or G_3) is of length $\lceil \log n' \rceil$, and thus $asat(R(\phi)) = m'(1 - 2^{-\lceil \log n' \rceil}) = m'(1 - \frac{2 \times 2^{-\lceil \log n' \rceil}}{2}) = m'(1 - \frac{2}{2^{\lceil \log n' \rceil + 1}}) = m'(1 - \frac{2}{m'}) = m' - 2$. So, if we fix the value of k to 2, Max-$r(n)$-SAT asks whether or not we can satisfy all the clauses of $R(\phi)$. To complete the proof, it must be shown ϕ is satisfiable if and only if $R(\phi)$ is satisfiable for $k = 2$.

Assume that there is an assignment A that satisfies ϕ. If we extend A to A' by assigning each $y_i, (1 \leq i \leq n' - n)$ variables the value $True$, then $R(\phi)$ is satisfied. To see this, note that A' satisfies G_1 and G_3 since every clause in them contains at least one positive literal. Also, A' satisfies G_2 since A satisfies every clause $C_i, (1 \leq i \leq m)$ in ϕ, which is included as part of each clause in G_2.

Suppose $R(\phi)$ is satisfied by some assignment. Then all the $y_i, (1 \leq i \leq \lceil \log n' \rceil)$ are set to $True$ in that assignment, or otherwise the clauses in C_1 having only one positive literal are not satisfied. Thus, the only literals satisfied in any clause belonging to G_2 are all the literals of $C_i, (1 \leq i \leq m)$, and hence ϕ is satisfied. □

Furthermore, Crowston *et al.* showed that

- Max-$r(n)$-SAT is not in XP for $r(n) \geq \log \log n + g(n)$, where $g(n)$ is any unbounded strictly increasing function.
- Max-$r(n)$-SAT is in XP for any $r(n) \leq \log \log n - \log \log \log n$.
- Max-$r(n)$-SAT is in FPT for any $r(n) \leq \log \log n - \log \log \log n - g(n)$, where $g(n)$ is any unbounded strictly increasing function.

3.4 Solving parameterized MaxSAT in $O^*(1.618^k)$ time

Bliznets and Golovnev[5] introduced an algorithm that solves parameterized MaxSAT in $O^*(1.358^k)$ time, where k is the parameter and O^* is the modified big-Oh notation introduced in[37] that suppresses all other polynomially bounded terms. This result is established by algorithm 4, whose running time and correctness are stated in this subsection. The previously known bounds are $O^*(1.618^k)$[21], $O^*(1.3995^k)$[26], $O^*(1.3803^k)$[4] and $O^*(1.3695^k)$[7]. In their paper, the following notations and definitions are used. Given a CNF formula ϕ with m clauses over n variables.

- MaxSAT$(\phi, k) = True$ if and only if we can satisfy at least k clauses in ϕ.
- $\phi[x = \neg y]$ denotes the formula obtained from ϕ by replacing x and $\neg x$ by $\neg y$ and y respectively.
- A variable x in ϕ has *degree* d, denoted $deg(x) = p$, if x occurs exactly d times in ϕ.
- A variable x in ϕ is said to be of *type* (a, b) if the literal x occurs a times and the literal $\neg x$ occurs b times.
- A variable x in ϕ is said to be $(k, 1)$-*singleton* ($(k, 1)$-*non-singleton*) if x is of type $(k, 1)$ and the only negation is contained (is not contained) in a unit clause.

- A literal l occurring in ϕ is called *pure* if the literal $\neg l$ does not appear in ϕ.
- A literal l_1 in ϕ is said to *dominate* another literal l_2 if all the clauses containing l_2 also contain l_1.
- A literal l_1 is said to be a *neighbor* to another literal l_2 if they appear together in a clause of ϕ.
- $\#_\phi(l)$ denotes the number of occurrences of a literal l in ϕ.
- For $q > 1$, it is said that there exists a *branching* $(a_1, \ldots, a_q)$ if we can construct formulae $\phi_1, \ldots, \phi_q$ such that the answer for the parameterized MaxSAT problem (ϕ, k) can be obtained from the answers for $(\phi_1, k - a_1), \ldots, (\phi_q, k - a_q)$. If l is a literal of ϕ, then clearly we can solve (ϕ, k) if we can solve $(\phi[l], k - \#_\phi(l)), (\phi[\neg l], k - \#_\phi(\neg l))$, and thus $(\#_\phi(l), \#_\phi(\neg l))$ is a branching.
- For a branching $(a_1, \ldots, a_q)$, where $a_i \leq a_j, (1 \leq i < j \leq q)$, a *branching number*, denoted by $\tau(a_1, \ldots, a_q)$, is the unique root[5] of the polynomial $X^{a_q} - (X^{a_q - a_1} + X^{a_q - a_2} + \cdots + X^{a_q - a_q})$. Theorem 2.1 in[19] suggests that if at each stage of an algorithm only branchings from the set $(a_{1,1}, \ldots, a_{1,q_1}), (q_{2,1}, \ldots, a_{2,q_2}), \ldots, (a_{t,1}, \ldots, a_{t,q_t})$ are used, where $a_{i,1} \leq a_{i,2} \leq \cdots \leq a_{i,q_i}, (1 \leq i \leq t)$, then the running time of the algorithm is $O^*(c^k)$, where c is the largest positive root of the polynomial
$$\prod_{j=1}^{t} \left(X^{a_{j,q_j}} - \sum_{i=1}^{q_j} X^{a_{j,q_j} - a_{j,i}} \right)$$
.
- A branching $(a_1, \ldots, a_q)$ dominates another branching $(b_1, \ldots, b_q)$ if for all $i = 1, \ldots, q$, we have $a_i \geq b_i$.

The authors presented algorithm 4, which establishes their new bound. The following simplification rules, lemmas and theorems are needed to prove the algorithm's correctness and running time.

Simplification rules

In the following rules, (ϕ, k) is a parameterized MaxSAT instance.

1. A literal l can be assigned the value $True$ if l is pure or if the number of unit clauses (l) is not smaller than the number of clauses containing $(\neg l)$.
2. A variable x with $deg(x) \leq 2$ can be eliminated. This is easy to see since at most one clause is falsified. To see this, consider th case that l is pure. If so, then l can be set to $True$. Otherwise, ϕ has the form $\phi' \cup \{(l \vee A), (\neg l \vee B)\}$ and $\text{MaxSAT}(\phi, k) = \text{MaxSAT}(\phi' \cup \{(A \vee B)\}, k - 1)$. Thus, the problem can be reduced.

[5] A root of a polynomial $P(X)$ is the number X_i such that $P(X_i) = 0$.

3. Pairs of clauses of the form (x) and $(\neg x)$ can be removed. This also easy to see because these two clauses form a contradiction and exactly one of them must be falsified. Thus, the problem is reduced and the parameter is decreased by one.

4. If two variables x and y with $deg(x) = deg(y) = 3$ appear together in three clauses, then these three clauses can be satisfied by assigning x and y. Indeed if both variables have degree three and they are neighbors in three clauses, then they certainly do not appear elsewhere and two out of three clauses can be satisfied by assigning x and the third can be satisfied by assigning y.

5. Let x be a variable with $deg(x) = 3$. Then ϕ is of the form $\phi' \cup \{(x \vee A), (x \vee B), (\neg x \vee C)\}$. If A or B have length less than two (either empty or a unit clause), then the problem (ϕ, k) can be reduced. That is, assume A is a unit clause, then it is easy to see that $\text{MaxSAT}(\phi, k) = \text{MaxSAT}(\phi' \cup \{(\neg A \vee B \vee C), (A \vee C)\}, k-1)$. If A is empty, then by setting x to $True$ we can decrease k by two. In both cases, the problem is reduced. The cases involving B are similar.

Each one of these simplification rules can be applied in polynomial time and whenever one of them is applied, the parameter k is decreased by at least one and the problem is reduced. Also, applying a rule satisfies come clauses.

Lemma 3.4. *If a CNF formula ϕ contains a variable x with $deg(x) \geq 6$, then the branching number on x is at most $\tau(1, 5)$.*

Proof. The possible branchings on x are $\tau(3, 3) = 1.25992$, $\tau(2, 4) = 1.27202$ and $\tau(1, 5) = 1.32472$. Clearly, $\tau(1, 5)$ is the largest branching number. □

Theorem 3.4. *If a variable x occurs three times in ϕ, then either the parameter can be decreased or there is a $(1, 6)$-, $(2, 4)$- or $(3, 3)$-branching.*

Theorem 3.5. *If all the variables in ϕ are either $(3, 1)$-singletons or $(4, 1)$-singletons, then at least $\lceil \frac{2m}{3} \rceil$ clauses of ϕ are satisfiable.*

Proof. Consider the formula $\psi = \{(\neg x), (\neg y), (\neg z), (x \vee y), (x \vee z), (x \vee y \vee z)\}$, where all the variables x, y and z are $(3, 1)$-singletons and $m = 7$. ψ is the smallest formula with 3 variables that are all $(3, 1)$ singletons and the maximum number of satisfied clauses in ψ is 5. Indeed, the assignment $A = \{x = True, y = False, z = True\}$ satisfies 5 clauses in ψ. Thus, at least $\lceil \frac{2m}{3} \rceil = \lceil 4.666 \rceil = 5$ clauses are satisfied. This also holds if we have $(3, 1)$-singletons and $(4, 1)$-singletons, in fact we can satisfy more. □

The minimum set cover (MSC) is a problem that is used in algorithm 4 to solve a special case of MaxSAT. The decision version of MSC is NP-complete and the optimization version is NP-hard.

Definition 3.8 (Minimum Set Cover (MSC))**.** *Given a universe U and a collection S of subsets of U, the MSC problem asks for the minimum cardinality of a subset S' of S such that $\bigcup_{S_i \in S'} S_i = U$. That is, the target is finding the minimum cardinality of a subset of S that covers U.*

If each set in S is of size at most four, van Rooij and Bodlaender developed an algorithm[35] that solves MSC in $O^*(1.29^{0.6|U|+0.9|S|})$ time, if each set in S has size at most four.

Algorithm 4: FPT-MaxSAT2(ϕ, k)

Input: An CNF formula ϕ with m clauses and a parameter k
Output: $True$, if there is an assignment that satisfies at least$k$ clauses of ϕ, otherwise $False$

1 apply simplification rules 1-5
2 **if** *there is a variable* x *with* $deg(x) \geq 6$ **then**
3 branch on x according to lemma 3.4
4 **if** *there is a variable* x *with* $deg(x) = 3$ **then**
5 branch on x according to theorem 3.4
  `// At this point, the remaining variables are of degree 4 and 5 only`
6 **if** *there is a variable* x *of type* $(3,2)$, $(3,1)$*-non-singleton or* $(4,1)$*-non-singleton* **then**
7 branch on x
  `// At this point, the remaining variables are either singletons or of type` $(2,2)$
8 **if** *there is a variable* x *of type* $(2,2)$ **then**
9 **if** x *has a neighbor* y *of type* $(4,1)$*-singleton and* x, $\neg x$ *are not simultaneously dominated by* y **then**
10 branch on y
11 **else**
12 branch on x
  `// At this point, all the variables are either` $(3,1)$`-singletons or` $(4,1)$`-singletons`
13 **if** $k \leq n$ **then**
14 **return** $True$
15 **if** $\frac{2m}{3} < k$ **then**
16 **if** $k \leq MSC(\phi)$ **then**
17 **return** $True$
18 **else**
19 **return** $False$
20 **if** *there is a clause of length 2,* $(x \vee y)$ **then**
21 branch as $\phi[x, y]$,$\phi[x = \neg y]$
22 **else**
23 **return** $True$

Algorithm 4 begins by simplifying the input formula according to the five simplification rules (line 1) stated previously. Next, variables of degrees greater than or equal to 6 are dealt with according to lemma 3.4 (lines 2-3), which suggests a $(1,5)$-branching. Thus, we have $\tau(1,5) = 1.32472 < 1.3579$. Variables of degrees 3 are dealt with according to theorem 3.4 (lines 4-6), which gives a $(1,6)$-branching. Thus, we have $\tau(1,6) = 1.2852 < 1.3579$. The remaining variables in ϕ now are of degrees 4 and 5. Lines 6-7 check the conditions where x is of type $(3,2)$, $(3,1)$-non-singleton or $(4,1)$-non-singleton. If x is of type $(3,2)$, then we get a $(3,2)$-branching, and so we have $\tau(3,2) = 1.32472 < 1.3579$. Else, x is a $(4,1)$-non-singleton or $(3,1)$-non-singleton and by branching on x we get at least $\tau(3,2) = 1.32472 < 1.3579$. Lines 8-10 handles the conditions of $(2,2)$ variables. If there is a $(2,2)$ variable x with a $(4,1)$-singleton y, such

that y does not dominate x and $\neg x$ together at the same time, then branching on y gives $\tau(4,1)$ and the next iteration in the $\phi[y]$ branch there is a variable of degree three or smaller. Thus, the total branching number is less than $\tau(4+1,4+6,1) = \tau(5,10,1) < 1.3579$. If the condition in line 9 is false, then x is of type $(2,2)$ and either the neighbors of x are variables of degree four, or x and $\neg x$ are simultaneously dominated by y. Thus, the two branches we get by branching on x (line 12) are $\phi[x]$ and $\phi[\neg x]$ contain a variable of degree three. By theorem 3.4, the possible branchings are $(1,6)$, $(2,4)$ and $(3,3)$, and so the possible branching numbers are $\tau(2+1,2+6,2+1,2+6)$, $\tau(2+2,2+3,2+2,2+3)$ and $\tau(2+3,2+3,2+3,2+3)$. The worst running time among these three corresponds to $\tau(3,8,3,8) \approx 1.3480 < 1.3579$.

At this point, the only variables remaining are of degree four and five, but singletons, specifically, $(3,1)$- and $(4,1)$-singletons. This means that all the negated literals appear in unit clauses. Thus, we n clauses can be satisfied by setting all the variables to $False$. If this satisfies the required number of clauses, then we are done (lines 13-14).

In line 15, the condition in the if-statement checks if k is greater than the guaranteed value proved by theorem 3.5. Note that all the clauses in ϕ are either unit clauses containing negated literals or clauses containing only positive literals. Indeed there always exists an optimal assignment that satisfies all the clauses containing only positive literals[6]. We want an assignment that assigns $True$ to the minimal number of variables and at the same time satisfies all the clauses containing no negated literals. This can be modeled as the MSC problem. The MSC instance is constructed as follows: U is the set of clauses containing no negated literals, $S = \{S_i \subset U \mid clausesin S_i containthevariable x_i\}$. Now, we want to cover U with the minimal number of sets from S. In line 16, the minimal number of subsets required to cover U is returned by the algorithm in[35]. IF t is the answer, then $m-t$ clauses can be satisfied (because t unit clauses are falsified). If $m-t$ is greater than or equal to k (line 16), the required number of satisfied clauses has been reached, if not then the algorithm terminates returning $False$ (line 19). Every set in S has cardinality at most four, since each variable appears positively in at most four clauses, and thus algorithm due to van Rooij and Bodlaender solves this MSC instance in $O^*1.29^{0.6|U|+0.9|S|}$. Since there are $m-n$ positive clauses, $|U| = m-n$. Also, since $n < k$ and $m < \frac{3k}{2}$, then the running time is $O^*(1.29^{1.2k})$, which is $\approx 1.3574^k < 1.3579^k$.

In line 20, there exists clauses $(\neg x)$ and $(\neg y)$ since all the variables are (a,b)-singletons. If there is a clause $(x \vee y)$, then it can be satisfied even if the optimal solution does not satisfy it. We can do this by setting x to $True$, which does not decrease the number of satisfied clauses. Thus, the algorithm branches as $x = y = True$ and $x = \neg y$. At least three clauses can be satisfied in the first branch since the variables are $(3,1)$- or $(4,1)$-singletons. In the second branch, by simplification rule 3 we can only satisfy one of (x) and $(\neg x)$ and also $(x \vee y)$ can be satisfied. Finally, the algorithm can $True$ in line 23 since $\frac{2m}{3} \geq k$.

[6]If a clause containing only positive literals is falsified, then we can flip any of the variables to satisfy that clause.

4 Conclusion

In this paper, we discussed the complexity of the Max-SAT problem in two ways: In classical computational complexity and parameterized complexity. The aim in both cases is to classify the problem into complexity classes via reductions. The main result in classical complexity is that the smallest s for which $(2, s)$-Max-SAT is in the NP-complete class is 3. The following two tables summarize the most important results discussed.

Problem	**Class**	**Reason**
Max-2-SAT	NP-complete	Red. from 3SAT[27]
$(\leq 2, 4)$-Max-SAT	NP-complete	Red. from vertex cover[29]
$(\leq 2, 3)$-Max-SAT	NP-complete	Red. from $(\leq 2, 4)$-Max-SAT[29]
$(2, 3)$-Max-SAT	NP-complete	Red. from $(\leq 2, 3)$-Max-SAT[29]
$(\leq n, 2)$-Max-SAT	P	Polynomial-time algorithm[29]

Table 1: Results in classical complexity.

Problem	**Class**	**Reason**
Parameterized MaxSAT	FPT	Algorithm 2
Satisfying at least $\lceil \frac{m}{2} \rceil + k$	FPT	Algorithm 3
Max-$r(n)$-SAT	para-NP-complete	Red. from Linear-3-SAT

Table 2: Results in parameterized complexity.

References

[1] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of mathematics*, pages 781–793, 2004.

[2] Roberto Asín Achá and Robert Nieuwenhuis. Curriculum-based course timetabling with sat and maxsat. *Annals of Operations Research*, pages 1–21, 2012.

[3] Mikhail J Atallah. *Algorithms and theory of computation handbook*. CRC press, 1998.

[4] Nikhil Bansal and Venkatesh Raman. Upper bounds for maxsat: Further improved. In *Algorithms and Computation*, pages 247–258. Springer, 1999.

[5] Ivan Bliznets and Alexander Golovnev. A new algorithm for parameterized max-sat. In DimitriosM. Thilikos and GerhardJ. Woeginger, editors, *Parameterized and Exact Computation*, volume 7535 of *Lecture Notes in Computer Science*, pages 37–48. Springer Berlin Heidelberg, 2012.

[6] Liming Cai and Jianer Chen. On fixed-parameter tractability and approximability of np optimization problems. *Journal of Computer and System Sciences*, 54(3):465–474, 1997.

[7] Jianer Chen and Iyad A Kanj. Improved exact algorithms for max-sat. In *LATIN 2002: Theoretical Informatics*, pages 341–355. Springer, 2002.

[8] Jianer Chen, Iyad A Kanj, and Ge Xia. Improved parameterized upper bounds for vertex cover. In *Mathematical Foundations of Computer Science 2006*, pages 238–249. Springer, 2006.

[9] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[10] Robert Crowston, Gregory Gutin, Mark Jones, Venkatesh Raman, and Saket Saurabh. Parameterized complexity of maxsat above average. *LATIN 2012: Theoretical Informatics*, pages 184–194, 2012.

[11] Pedro Filipe Medeiros da Silva. Max-sat algorithms for real world instances. 2010.

[12] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[13] Michael R Garey, David S Johnson, and Larry Stockmeyer. Some simplified np-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63. ACM, 1974.

[14] Michael R Garey, David S. Johnson, and Larry Stockmeyer. Some simplified np-complete graph problems. *Theoretical computer science*, 1(3):237–267, 1976.

[15] Gregory Gutin, Mark Jones, and Anders Yeo. A new bound for 3-satisfiable maxsat and its algorithmic application. In *Fundamentals of Computation Theory*, pages 138–147, 2011.

[16] Mikoláš Janota, Inês Lynce, Vasco Manquinho, and Joao Marques-Silva. Packup: Tools for package upgradability solving system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 8:89–94, 2012.

[17] Brigitte Jaumard and Bruno Simeone. On the complexity of the maximum satisfiability problem for horn formulas. *Information Processing Letters*, 26(1):1–4, 1987.

[18] RichardM. Karp. Reducibility among combinatorial problems. In RaymondE. Miller, JamesW. Thatcher, and JeanD. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972.

[19] Dieter Kratsch. *Exact Exponential Algorithms*. Springer-Verlag Berlin Heidelberg, 2010.

[20] Karl J Lieberherr and Ernst Specker. Complexity of partial satisfaction. *Journal of ACM (JACM)*, 28(2):411–421, 1981.

[21] Meena Mahajan and Venkatesh Raman. Parameterizing above guaranteed values: Maxsat and maxcut. *Journal of Algorithms*, 31(2):335–354, 1999.

[22] Filip Maric. Timetabling based on sat encoding: a case study, 2008.

[23] Elizabeth Montero, María-Cristina Riff, and Leopoldo Altamirano. A pso algorithm to solve a real course+ exam timetabling problem. In *International Conference on Swarm Intelligence*, pages 24–1, 2001.

[24] Fahima NADER, Mouloud KOUDIL, Karima BENATCHBA, Lotfi ADMANE, Said GHAROUT, and Nacer HAMANI. Application of satisfiability algorithms to time-table problems. *Rapport Interne LMCS, INI*, 2004.

[25] G-J Nam, Fadi Aloul, Karem A. Sakallah, and Rob A. Rutenbar. A comparative study of two boolean formulations of fpga detailed routing constraints. *Computers, IEEE Transactions on*, 53(6):688–696, 2004.

[26] Rolf Niedermeier and Peter Rossmanith. *New upper bounds for MaxSat*. Springer, 1999.

[27] Christos H Papadimitriou. *Computational complexity*, chapter 9: NP-complete Problems. Addison-Wesley, 1994.

[28] Wayne Pullan. Protein structure alignment using maximum cliques and local search. In *AI 2007: Advances in Artificial Intelligence*, pages 776–780. Springer, 2007.

[29] Venkatesh Raman, Bala Ravikumar, and S Srinivasa Rao. A simplified np-complete maxsat problem. *Information Processing Letters*, 65(1):1–6, 1998.

[30] Sean Safarpour, Hratch Mangassarian, Andreas Veneris, Mark H Liffiton, and Karem A Sakallah. Improved design debugging using maximum satisfiability. In *Formal Methods in Computer Aided Design, 2007. FMCAD'07*, pages 13–19. IEEE, 2007.

[31] Tian Sang, Paul Beame, and Henry Kautz. A dynamic approach to mpe and weighted max-sat. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 173–179. Morgan Kaufmann Publishers Inc., 2007.

[32] Craig A Tovey. A simplified np-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984.

[33] Craig A. Tovey. A simplified np-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85 – 89, 1984.

[34] Jan Van Leeuwen and Jan Leeuwen. *Handbook of theoretical computer science: Algorithms and complexity*, volume 1, chapter 2: A Catalog of Complexity Classes. Elsevier, 1990.

[35] Johan MM Van Rooij and Hans L Bodlaender. Exact algorithms for dominating set. *Discrete Applied Mathematics*, 159(17):2147–2164, 2011.

[36] Michel Vasquez and Jin-Kao Hao. A logic-constrained knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. *Computational Optimization and Applications*, 20(2):137–157, 2001.

[37] GerhardJ. Woeginger. Exact algorithms for np-hard problems: A survey. In Michael Jnger, Gerhard Reinelt, and Giovanni Rinaldi, editors, *Combinatorial Optimization Eureka, You Shrink!*, volume 2570 of *Lecture Notes in Computer Science*, pages 185–207. Springer Berlin Heidelberg, 2003.

[38] Hui Xu, Rob A Rutenbar, and Karem Sakallah. sub-sat: a formulation for relaxed boolean satisfiability with applications in routing. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 22(6):814–820, 2003.